Available online at www.sciencedirect.com

ScienceDirect

Journal of Discrete Algorithms 6 (2008) 93–102

JOURNAL OF DISCRETE ALGORITHMS

www.elsevier.com/locate/jda

Generating all cycles, chordless cycles, and Hamiltonian cycles with the principle of exclusion

Marcel Wild

Department of Mathematical Sciences, University of Stellenbosch, Van der Ster Gebou, kamer 2023, 7602 Stellenbosch, South Africa

Received 24 January 2006; received in revised form 4 January 2007; accepted 8 January 2007

Available online 27 April 2007

Abstract

An efficient method to generate all edge sets $X \subseteq E$ of a graph $G = (V, E)$, which are vertex-disjoint unions of cycles, is presented. It can be tweaked to generate (i) all cycles, (ii) all cycles of cardinality $\leqslant 5$, (iii) all chordless cycles, (iv) all Hamiltonian cycles.

Keywords: Generating all Hamiltonian cycles; Generating all cycles; Generating all chordless cycles

1. Introduction

The core of this article is an efficient method to generate all Hamiltonian cycles of an arbitrary connected graph. The Mathematica code (available upon request) to generate them all beats the hardwired Mathematica command `HamiltonianCycle[G,All]` by a factor fifty on a random graph with 30 vertices, 67 edges, and 35840 Hamiltonian cycles. Our method is based on a novel way to produce all vertex-disjoint unions of cycles (e.g. single cycles) of a graph. Correspondingly the latter problem, which is of interest in itself, will be treated first. Underlying both is what we call the principle of exclusion. The next two paragraphs describe that principle in a quite general setting, thereby slightly raising the usual level of technicality encountered in an introduction. This is necessary anyway, and will allow for a more coherent outline of the section break up of the article.

The *principle of exclusion* generates all N combinatorial objects X of a given type, provided these objects can be defined by "local" properties $\mathcal{P}_i$ in a wide sense. As a bonus, this method is conveniently structured in table form, and invites hand calculations. Formally, a *property* of subsets of a fixed set E is a family $\mathcal{P}_i \subseteq 2^E$, and $X \subseteq E$ *has* property $\mathcal{P}_i$ if and only if $X \in \mathcal{P}_i$. Usually the families $\mathcal{P}_i$ are way too large to store explicitly, yet the task to find all $X \subseteq E$ satisfying v given properties $\mathcal{P}_i$ amounts to determine $\mathcal{P}_1 \cap \mathcal{P}_2 \cap \cdots \cap \mathcal{P}_v$. Our basic idea to achieve this is disarmingly simple. Rather than starting with the empty set and adding the N objects one by one, we begin with the powerset $C_0 := 2^E$ and *exclude* the non-objects. More specifically, $C_{i+1} := C_i \cap \mathcal{P}_{i+1}$ arises from C_i by excluding all $X \in C_i$ that fail to have properly $\mathcal{P}_{i+1}$. Thus after v steps we arrive at $C_v = \mathcal{P}_1 \cap \cdots \cap \mathcal{P}_v$. This looks like a naive approach but a compact way of representing C_0 up to C_v makes it work.

E-mail address: mwild@sun.ac.za.

1570-8667/$ – see front matter
doi:10.1016/j.jda.2007.01.005

It is best to give some more details right away. Each $X \in C_i$ will be identified with its characteristic 0, 1-vector of length $w = |E|$, which we shall refer to as a *row*. Whenever possible we use the symbol 2 to indicate that an entry is freely allowed to be 0 or 1, as if it was in a quantum superposition! For instance $C_i = \{(2, 2, 0, 1, 0, 2)\}$ (never mind the curly brackets) comprises $2^3 = 8$ objects, one of them is e.g. $(1, 0, 0, 1, 0, 1)$. Say the follow up $C_{i+1} = \{(0, 2, 0, 1, 0, 2), (1, 2, 0, 1, 0, 1)\}$ consists of two sons, i.e. *quantum rows* whose disjoint union comprises $4 + 2 = 6$ objects. In fact, they are the X's in C_i satisfying the property $\mathcal{P}_{i+1}$ that $(1 \in X \Rightarrow 6 \in X)$. Generally the elements of C_i can be viewed as tree-nodes which, when deleted, altered or split, yield the elements of C_{i+1}. If an intermediate *context* C_i has gotten too large, one can start to "finalize" its rows r one by one, because the future sons and grandsons of r are independent of $C_i \setminus \{r\}$. Thus, no space problem arises. Besides 0, 1, 2, any particular variant of the principle of exclusion utilises its own specific symbols; in the present article d, f, g. For instance, $(g, g, \ldots, g)$ abbreviates $\{(1, 0, \ldots, 0), (0, 1, 0, \ldots, 0), \ldots, (0, \ldots, 0, 1)\}$. The principle of exclusion runs faster if the expensive deletion of rows is avoided. In certain situations this can be guaranteed, in our situation not. But suitable precautions will prevent an excessive deletion of rows, thereby keeping the principle of exclusion efficient.

Here comes the section break up. Throughout, $G = (V, E)$ will be a simple connected graph with vertex set V and edge set E. Put $v = |V|$ and $w = |E|$. The key idea is to look at *vertex*-disjoint unions $X \subseteq E$ of cycles, rather than edge-disjoint unions. The latter enjoy a vector space structure but are far more numerous. More importantly, the principle of exclusion fits the former like a glove since neat local edge set properties $\mathcal{P}_i$, corresponding to the vertices α_i, suggest themselves (Section 2). Picking those X's that happen to be single cycles works well enough (Section 3). Notice that a vertex-disjoint union X of cycles with $|X| \leqslant 5$ necessarily *is* a cycle. The effect is that all the cycles of cardinality at most five can be produced particularly fast. It is also easy to adapt our algorithm so as to produce only the chordless cycles (Section 4). By definition, a vertex-disjoint *partition* $X \subseteq E$ of cycles covers all of V. Again these types of objects X are predestined for the principle of exclusion. Those X's that happen to be single cycles, are necessarily Hamiltonian cycles (Section 5). Finally, Hamiltonian paths are dealt with in Section 6. Throughout the article our algorithms will be illustrated on a concrete graph G with $v = 6$ and $w = 10$.

2. Vertex-disjoint cycle packings

First we review how people used edge-disjoint cycle packings in order to generate all cycles of a graph. We then proceed to argue that vertex-disjoint cycle packings are fitter for the task: They lack the algebraic niceties of the former but conform better to the principle of exclusion.

The most tempting approach to generate all cycles of a connected graph is as follows. Call $X \subseteq E$ an *ed cycle packing* if it is an edge-disjoint union of cycles (perhaps a single cycle or the empty set). It is well known [6] that the set P of all ed cycle packings is a $\mathbb{Z}_2$-vector space with symmetric difference of sets as addition. The dimension of P is $w - v + 1$ and a base B of P is e.g. given by taking all *fundamental cycles* w.r.t. a fixed spanning tree of G. Hence B is easily found, P generated from B in a straightforward manner, and the inclusion-minimal $X \in P$ are precisely the cycles. This approach has been refined in [8] and [1], and for planar graphs in [7], but it suffers from the fact that the number of 2^{v-w+1} ed cycle packings is forbidding. This led to a variety of backtracking algorithms, surveyed in [5]. Some of these do the job in time $O(v + w + wN)$ where N is the number of cycles. See also [4].

Our approach to generate all sorts of cycles comes without performance guarantee but plenty of experiments on random graphs speak another language. Although we stick to cycle packings we focus on the far fewer *vertex-disjoint* (vd) cycle packings which are defined in the obvious way. To be concise, for any subset X of edges and any vertex α define the *degree of α in X* as

$$\deg(X, \alpha) := \big|\mathrm{star}(\alpha) \cap X\big|,$$

where $\mathrm{star}(\alpha)$ is the set of edges incident with α. Thus, $X \subseteq E$ is an ed cycle packing iff

$$(\forall \alpha \in V) \quad \deg(X, \alpha) \equiv 0 \pmod 2,$$

and it is a vd cycle packing iff

$$(\forall \alpha \in V) \quad \deg(X, \alpha) \in \{0, 2\}. \tag{1}$$

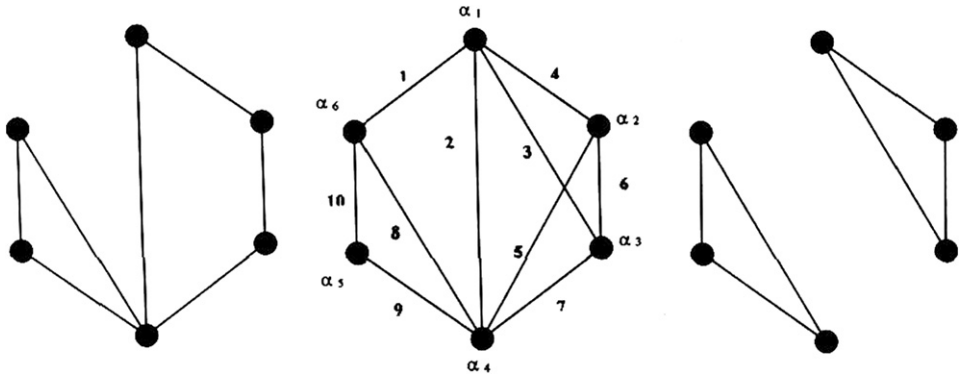

Fig. 1.

Condition (1) is localised to single vertices and whence well suited to the principle of exclusion. Thus, putting $V = \{\alpha_1, \ldots, \alpha_v\}$, by definition $X \subseteq E$ has property $\mathcal{P}_i$ (see introduction) if $\deg(X, \alpha_i) \in \{0, 2\}$. Consider the $(6, 10)$-graph G which has $2^{10-6+1} = 32$ ed cycle packings, one of them shown to the left of G (see Fig. 1).

In order to find the fewer *vd* cycle packings, one of them shown to the right of G, we employ quantum rows as outlined in the introduction, and besides $0, 1, 2$ introduce two new symbols. Namely, write $ff \ldots f$ if either all symbols f are 0, or exactly two symbols f are 1 and the others 0. Similarly write $gg \ldots g$ if exactly one symbol g is 1 and the others 0. To use the suggestive (if ridiculous) quantum picture: Here 1 is a wave rather than a particle. The family of all $X \subseteq E$ with $\deg(X, \alpha_1) \in \{0, 2\}$ is therefore the row r in Table 1 which has its components indexed by the edges $1, 2, \ldots, 10$ of G.

Before we can impose on it the constraint $\deg(X, \alpha_2) \in \{0, 2\}$, we must pin down the entry in $\text{star}(\alpha_1) \cap \text{star}(\alpha_2) = \{4\}$. Trivially r is the disjoint union of $r_1(-) := \{X \in r\colon\ 4 \notin X\}$ and $r_2(-) := \{X \in r\colon\ 4 \in X\}$. In our new notation

$$r_1(-) = (f, f, f, \mathbf{0}, 2, 2, 2, 2, 2, 2),$$
$$r_2(-) = (g, g, g, \mathbf{1}, 2, 2, 2, 2, 2, 2).$$

The constraint $\deg(X, \alpha_2) \in \{0, 2\}$ is now smoothly imposed on both $r_1(-)$ and $r_2(-)$ because the sets $r_1 := \{X \in r_1(-)\colon \deg(X, \alpha_2) \in \{0, 2\}\}$ and $r_2 := \{X \in r_2(-)\colon \deg(X, \alpha_2) \in \{0, 2\}\}$ fit nicely the f, g-mechanism. They constitute the intermediate context $C_2 = \{r_1, r_2\}$.

In general, imposing the $(i + 1)$th constraint on a row $\rho \in C_i$ either causes local changes in ρ and results in one *unsplit son* ρ_1, or causes ρ to split into at least two *candidate sons* ρ_k. Each ρ_k, unsplit or not, undergoes a subroutine `RowFutureBleakQ`$[\rho_k]$ that glimpses at the upcoming vertices α_j $(i + 2 \leqslant j \leqslant v)$. If the outcome is `True`, ρ_k is cancelled. If it is `False`, ρ_k is promoted, possibly in altered form, to *proper son* $\rho_k(+)$.

For instance, in order to process α_3 first note that $\text{star}(\alpha_3) = \{3, 6, 7\}$ "covers" the symbols f_1 and f_2 in $\rho := r_1$. Hence these two f_1 and f_2 are chosen from $\{0, 1\}$ in $2^2 = 4$ ways (indicated boldface), giving rise to four candidate

Table 1

	1	2	3	4	5	6	7	8	9	10	
r	f	f	f	f	2	2	2	2	2	2	C_1
r_1	f_1	f_1	f_1	0	f_2	f_2	2	2	2	2	C_2
r_2	g_1	g_1	g_1	1	g_2	g_2	2	2	2	2	
ρ_1	f	f	**0**	0	0	**0**	0	2	2	2	
ρ_2	f	f	**0**	0	1	**1**	1	2	2	2	
ρ_3	g	g	**1**	0	0	**0**	1	2	2	2	
ρ_4	g	g	**1**	0	1	**1**	0	2	2	2	
	g	g	**0**	1	1	**0**	0	2	2	2	C_3'
	g	g	**0**	1	0	**1**	1	2	2	2	
	0	0	**1**	1	1	**0**	1	2	2	2	
	0	0	**1**	1	0	**1**	0	2	2	2	

sons ρ_1 to ρ_4. More precisely, setting $f_2 = \mathbf{0}$ in position 6 of ρ forces $f_2 = 0$ in position 5. Similarly, setting $f_1 = \mathbf{0}$ in position 3 of ρ clearly changes $f_1 f_1 f_1$ to $f_1 f_1 0$. Now we are in a position to impose the constraint $\deg(X, \alpha_3) \in \{0, 2\}$ upon our *precandidate son*

$$\rho_1(-) := (f_1, f_1, \mathbf{0}, 0, 0, \mathbf{0}, 2, 2, 2, 2).$$

Because of the 0's in position 3 and 6 only $\deg(X, \alpha_3) = 0$ can be forced, namely by switching the symbol 2 in position 7 to 0. The resulting row ρ_1 is the first candidate son of ρ (the $f_1 f_1$ are rewritten as ff in Table 1). Here $\texttt{RowFutureBleakQ}[\rho_1] = \texttt{False}$, and the proper son $\rho_1(+)$ coincides with ρ_1 since the upcoming vertices α_j ($4 \leqslant j \leqslant 6$) present no problems. The second candidate son ρ_2, induced by setting $f_1 = \mathbf{0}$, $f_2 = \mathbf{1}$ in ρ, is more stubborn. Namely, since ρ_2 has 1's on the positions 5, 7 of $\text{star}(\alpha_4) = \{2, 5, 7, 8, 9\}$, the elements on the positions 2, 8, 9 are switched to 0. But along with the f in second position, the f in first position must be turned to 0 as well:

$$(\mathbf{0}, \mathbf{0}, 0, 0, 1, 1, 1, \mathbf{0}, \mathbf{0}, 2).$$

(Had there been gg at the beginning of ρ_2, turning the second g to 0 would have triggered the first g to turn to 1.) Iterating this process of 0, 1-*propagation* we also turn the last symbol 2 to 0 since $\text{star}(\alpha_5) = \{9, 10\}$. (Had the last symbol been a 1, the outcome would have been $\texttt{RowFutureBleakQ}[\rho_2] = \texttt{True}$). Because $\text{star}(\alpha_6) = \{1, 8, 10\}$ is covered by 0's, it triggers no changes. Thus we get again $\texttt{RowFutureBleakQ}[\rho_2] = \texttt{False}$, but ρ_2 must[1] change to

$$\rho_2(+) := (0, 0, 0, 0, 1, 1, 1, 0, 0, 0).$$

In general the rounds of 0, 1-propagation are repeated until a round with no changes occurs. Applying RowFutureBleakQ, the reader is invited to fill in how the collection C_3' of candidate sons of r_1 and r_2 transforms to the intermediate context C_3. Continuing in this fashion one gets C_4, C_5, and C_6. In fact, $C_6 = C_5$ since always $C_v = C_{v-1}$ (why?).

Henceforth *cycle packing means vd cycle packing*. A moment's thought shows that a *final context* C_v of cycle packings necessarily[2] consists of 0, 1-rows. In our example there is only one out of nineteen nonempty cycle packings which is not a cycle (namely the one shown in Fig. 1).

Here is some systematic notation. Put $E = \{1, 2, \ldots, w\}$ for simplicity. For a given row r let $\text{zeros}(r) \subseteq E$ be the set of *positions* on which r carries a 0. Similarly define $\text{ones}(r)$ and $\text{twos}(r)$. Furthermore, let $\text{Con}(r, f)$ be the set of all position sets of f-constraints. Similarly $\text{Con}(r, g)$ is defined. Put $\text{Con}(r) := \text{Con}(r, f) \cup \text{Con}(r, g)$ and $\text{con}(r) := E \setminus (\text{zeros}(r) \cup \text{ones}(r) \cup \text{twos}(r))$. Finally set

$$w_{\min}(r) := \min\bigl\{|X|\colon\ X \in r\bigr\} = \bigl|\text{ones}(r)\bigr| + \bigl|\text{Con}(r, g)\bigr|,$$
$$w_{\max}(r) := \max\bigl\{|X|\colon\ X \in r\bigr\} = w_{\min}(r) + \bigl|\text{twos}(r)\bigr| + 2\bigl|\text{Con}(r, f)\bigr|.$$

For instance, the quantum row

$$r := (2, g_1, 0, 1, g_1, 0, g_1, f, f, 0, g_2, f, g_2, g_2)$$

has

$$\text{zeros}(r) = \{3, 6, 10\}, \qquad \text{ones}(r) = \{4\}, \qquad \text{twos}(r) = \{1\}$$
$$\text{Con}(r, g) = \bigl\{\{2, 5, 7\}, \{11, 13, 14\}\bigr\}, \qquad \text{Con}(r, f) = \bigl\{\{8, 9, 12\}\bigr\}$$
$$\text{con}(r) = \{2, 5, 7, 8, 9, 11, 12, 13, 14\}$$
$$w_{\min}(r) = 1 + 2 = 3,$$
$$w_{\max}(r) = 3 + 1 + 2 \cdot 1 = 6,$$

and that's exactly how quantum rows are formally defined and implemented in our Mathematica code.

[1] Strictly speaking ρ_2 need not change; all that matters is to know that at least one descendant of ρ_2 will survive to the final context. However, filling in 0's and 1's right away decreases the number of splittings that lie ahead.

[2] As opposed to cycle packings or perfect matchings [9], a final context of hypergraph transversals or of order ideals, needs not consist of 0, 1-rows (research in progress).

Let us recap the general procedure, call it (f, g)-*algorithm*, that produces all vdcpack(G) many nonempty cycle packings of a connected (v, w)-graph. We process the vertices α_i of G in any order.[3] For given $r \in C_i$ we impose the constraint $\deg(X, \alpha_{i+1}) \in \{0, 2\}$ via our devices $f \dots f$ and $g \dots g$. More specifically, either r has an unsplit son, or $1 + s + \binom{s}{2}$ candidate sons r_k. Here s is the number of positions (= elements) in $\text{star}(\alpha_{i+1}) \cap \text{con}(r)$, positions on which we attempt to write 0's and 1's in all manners that feature exactly zero, one, or two 1's. Depending on whether a 0 or 1 is put on a f-constraint or g-constraint, the latter change or vanish in obvious ways. As seen, some candidate sons are rejected by the `RowFutureBleakQ` test, the others are made proper sons. We postpone experimental data to Sections 4 and 6. The pseudocode of a similar (d, g)-algorithm will be displayed in Section 6.

3. All cycles

How can we adapt the (f, g)-algorithm so as to discard all *proper* cycle packings (i.e. comprising at least two cycles) and only keep the cycle packings that consist of single cycles? Throughout the algorithm for any row r the subgraph S induced by the edge set $\text{ones}(r)$ trivially is a vertex-disjoint union of paths and cycles. Picking a random edge $x \in \text{ones}(r)$ and iteratively checking for adjacent edges in S one can decide fast[4] whether or not this property takes place:

$$\text{ones}(r) \text{ contains a cycle but is not itself a cycle.} \tag{2}$$

If the answer is "yes", row r must be cancelled because all its descendants would be proper cycle packings. An answer "no" is inconclusive; perhaps row r or some of its descendants yield "yes" later on. Call any edge set that *properly* contains a cycle a *cancerous cycle*. Thus, enhancing the (f, g)-algorithm with the above described subroutine `CancerousCycleQ`$[r]$ provides us with all the $\text{cyc}(G)$ many cycles of G.

4. All small cycles and all chordless cycles

Notice that a cycle packing X with $|X| \leqslant 5$ is necessarily a cycle. This immediately leads to the $(f, g, 5)$-*algorithm* that produces, without bothering about `CancerousCycleQ`, all the $\text{cyc}_5(G)$ many cycles of cardinality at most five.

Trickier is our task in the remainder of this section, namely to generate exactly all *chordless* cycles $X \subseteq E$ (i.e. X is a cycle such that $E - X$ has no edges between vertices on X). For that purpose, given $x \in E$, say $x = \{\alpha, \beta\}$, define

$$\text{starpair}(x) := \big(\text{star}(\alpha) \cup \text{star}(\beta)\big) \setminus \{x\}.$$

Suppose that during the (f, g)-algorithm a row r arises such that

$$\big|\text{ones}(r) \cap \text{starpair}(x)\big| \geqslant 3 \tag{3}$$

for some $x \in E$. Pictorially it is shown in Fig. 2.

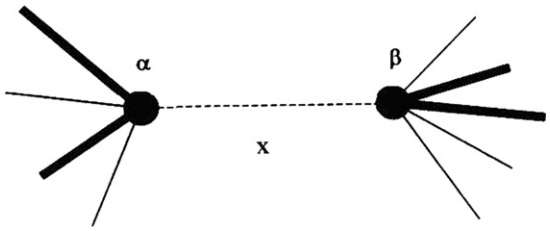

Fig. 2.

[3] Whether any particular kind of vertex ordering is advantageous, remains an open question. In other applications of the principle of exclusion (e.g. order ideals of a poset) the ordering is well important.

[4] Using the hardwired commands `ConnectedComponents` and `TreeQ` was marginally faster. However, one can improve upon `ConnectedComponents` plus `TreeQ` by storing only the *endpoints* $\{\alpha_i, \beta_i\}$ of all paths $P_i \subseteq \text{ones}(r)$, and updating them, e.g. joining them, as new edges x are added. Obviously $x = \{\alpha, \beta\}$ completes a cycle if and only if $\{\alpha, \beta\} = \{\alpha_i, \beta_i\}$ for some i. Disregarding the overhead, the improvement is the better the bigger the task (e.g. 30% gain on 4 minutes).

Table 2

(v, w)	$\text{cyc}_5(G)$		$\text{clcyc}(G)$		$\text{cyc}(G)$		$\text{vdcpack}(G)$	
(10, 23)	122	9	28	36	1103	2	1571	0
(10, 34)	910	0	66	168	39123	9	56149	0
(30, 67)	128	169	1464	8003				
(50, 77)	41	41	700	1634	824483	4181		

Notice that no three fat edges can ever be incident with α (or all with β) because either the constraint $\deg(X, \alpha) \in \{0, 2\}$ has been already imposed, or `RowFutureBleakQ` has looked ahead to prevent a threefold incidence. From the picture it is clear[5] that each $Y \in r$ that happens to be a cycle, must have the chord x, and so r must be cancelled. We can get all the $\text{clcyc}(G)$ many chordless cycles of G by augmenting the (f, g)-algorithm with a subroutine `ChordyCycleQ`$[r]$ that outputs `True` if (3) takes place for some $x \in E$, and `False` otherwise.

Chordless cycles can e.g. be used in connection with balanced matrices [2], or to compute the flat lattice $\mathcal{L}(E)$ of the matroid on the edge set of a graph (since $F \subseteq E$ is a flat iff $|X \cap F| \geqslant |X| - 1$ implies $X \subseteq F$, for all chordless cycles X).

Table 2 contains some numerical data about our (f, g)-algorithm, and its discussed adaptions. Recall that $v =$ number of vertices, $w =$ number of edges, of our random (connected) graph G. It strikes one that the number $\text{vdcpack}(G)$ of (vd) cycle packings of G is always within a factor two of $\text{cyc}(G)$, e.g. $56149 < 2 \cdot 39123$, whereas the number of *edge*-disjoint cycle packing e.g. soars to $2^{25} = 33554432$ for the $(10, 34)$-graph. Notice the tiny fraction of chordless cycles, even for sparse graphs. In lack of competitors[6] the execution times would not be very meaningful. More relevant is the number of *harmful* deletions (second entry), i.e. the number of times a quantum row was killed because *none* of its candidate sons survived. For cycles and cycle packings the proportion $\text{hd}(G)$ of harmful deletions w.r.t. $|C_v|$ was very low. In fact, even the proportion of *all* deleted rows r (due to `RowFutureBleakQ`$[r] =$ `True` or `CancerousCycleQ`$[r] =$ `True`) was low. For instance, it took 3777 sec to compute the 824 483 cycles of the $(50, 77)$-graph in Table 2; 4181 rows were killed harmfully, and 517 647 rows were killed altogether. For chordless cycles and cycles with at most five elements $\text{hd}(G)$ increased considerably, to the extent that sometimes intermediate context lengths $|C_i|$ were larger than the final number $|C_v|$ of objects. Nevertheless, the execution times dropped a lot w.r.t. the times for $\text{cyc}(G)$.

5. All Hamiltonian cycles

In Section 2 we introduced the (f, g)-algorithm that generates all cycle packings of a connected graph. In Section 3 we used `CancerousCycleQ` in order to only keep the cycles. In this section, in order to get only *Hamiltonian* cycles, we shall use the stronger watchdog `SmallCycleQ` that also discards all "healthy" cycles of cardinality $< v$. Furthermore, rather than dealing with arbitrary cycle packings, it will be more efficient to go for the significantly fewer cycle *partitions* (= cycle packings of size v) right from the start. After introducing a new $dd \ldots d$ symbolism that is taylored to cycle partitions, we apply the (d, g)-algorithm to obtain the Hamiltonian cycles of the graph in Fig. 1. We then give a pseudocode of the (d, g)-algorithm, followed by some experimental data.

Formally, a subset $X \subseteq E$ is a *cycle partition* iff[7]

$$(\forall \alpha \in V) \quad \deg(X, \alpha) = 2. \tag{4}$$

Thus a cycle partition is a (vertex-) disjoint union of cycles covering all of V, and so the Hamiltonian cycles are precisely the *connected* cycle partitions. Unfortunately connectedness cannot be ensured by degree conditions, but we

[5] This is false if $|\text{ones}(r) \cap \text{starpair}(x)| = 2$ since x could be *part* of cycle Y. Also observe that equality in (3) entails that r contains *no cycle at all*.

[6] The author was unable to obtain a Mathematica code of one of the backtracking algorithms of [5], but presumably they compare to the (f, g)-algorithm no better than the backtracking based `HamiltonianCycle[G, All]` in the next section.

[7] A Hamiltonian cycle is a particular kind of *edge cover*. Spanning trees and perfect matchings are other kinds. The latter can be defined similarly to (4) by $(\forall \alpha \in V)\ \deg(X, \alpha) = 1$. What we call the *g-algorithm* in [9] produces all perfect matchings of a (not necessarily bipartite) graph.

Table 3

	1	2	3	4	5	6	7	8	9	10	
	d	d	d	d	2	2	2	2	2	2	C_1
r	d	d	d	0	1	1	2	2	2	2	C_2
$\bar{r}$	g_1	g_1	g_1	1	g_2	g_2	2	2	2	2	
r_1	1	1	**0**	0	1	1	1	2	2	2	
r_2	g	g	**1**	0	1	1	0	2	2	2	
$\bar{r}_1$	g	g	**0**	1	0	**1**	1	2	2	2	C'_3
$\bar{r}_2$	0	0	**1**	1	1	**0**	1	2	2	2	
$\bar{r}_3$	0	0	**1**	1	0	**1**	0	2	2	2	
s_1	1	**0**	1	0	1	1	0	g	g	2	
s_2	0	**1**	1	0	1	1	0	0	0	2	C'_4
t_1	1	**0**	0	1	0	1	1	g	g	2	
t_2	0	**1**	0	1	0	1	1	0	0	2	
	1	0	1	0	1	1	0	0	1	1	$C_5 = C_6$
	1	0	0	1	0	1	1	0	1	1	

shall get around that obstacle[8] pretty well. We will write $dd\ldots d$ to indicate that exactly two symbols d are 1 and the others 0.

Let us generate all Hamiltonian cycles of our friend G from Fig. 1 with the (d, g)-*algorithm* in a manner akin to Section 2 (see Table 3).

It should be clear how $C_2 = \{r, \bar{r}\}$ arises from C_1 upon imposing the constraint $\deg(X, \alpha_2) = 2$, and how the respective candidate sons (collected in C'_3) arise upon imposing $\deg(X, \alpha_3) = 2$. Now r_1 must be cancelled because $\text{ones}(r_1)$ contains the cycle $\{5, 6, 7\}$ of cardinality $< v$ (thus `SmallCycleQ`$[r_1] =$ `True`). By analogous reasons $\bar{r}_2$ and $\bar{r}_3$ are cancelled. From the remaining context $C_3 = \{s_1, s_2\} := \{r_2, \bar{r}_1\}$ we get C'_4 upon imposing the constraint $\deg(X, \alpha_4) = 2$. But s_2, t_2 must be deleted because of $w_{\max}(s_2)$, $w_{\max}(t_2) < v$. Therefore C_4 comprises s_1, t_1, and each contributes one proper son to C_5. As previously $C_v = C_{v-1}$, and so $C_6 = C_5$. The rows of C_6 are the Hamiltonian cycles $\{1, 3, 5, 6, 9, 10\}$ and $\{1, 4, 6, 7, 9, 10\}$ of G, and they must be the only ones. In general, the subroutine `RowFutureBleakQ` is more active; see the example in the next section.

Since the pseudocode of the (d, g)-algorithm (Table 4) is akin to the Mathematica programming language, let us first outline its handling of `Do`, `While`, and `If`, since this may be unfamiliar to some readers:

- `Do[expr, {`$i, i_{\max}$`}]` evaluates `expr` with the variable i running from 1 to $i_{\max}$.
- `While[test, body]` evaluates `test`, then `body`, repetitively, until `test` first fails to give `True`.
- `If[condition,t, f]` gives `t` if `condition` evaluates to `True`, and `f` if it evaluates to `False`.
- `If[condition,t]` gives `t` if `condition` evaluates to `True`; otherwise the next command is executed.
- `(* Comments *)` ignores `Comments`.

In the example above, 0, 1-propagation was incidentally not an issue. In general, the opposite is true. For better readability, in Table 4 the subroutine `RowFutureBleakQ` is meant to comprise three ingredients: 0, 1-propagation as usual, but also the tests `SmallCycleQ` and "$w_{\max} < v$?". We chose to write ¬`RowFutureBleakQ`$[\ldots]$ since this looks better than the equivalent `RowFutureBleakQ`$[\ldots] ==$ `False`. Not surprisingly, the placement of quantum rows within a context C_i is more systematic now, than it was in Table 3. By induction, once the computation of C_i is achieved, the dynamic variable c equals $|C_i|$ (that is, the number of rows contained in C_i). In order to get C_{i+1} a pointer p decreases from $p = c$ to $p = 0$ (see the `While`-loop). In the interesting case where `row`$[p]$ sparks candidate sons, the following happens. Having saved `row`$[p]$ as `Bygone`, `row`$[p]$ is kicked and `row`$[c]$ takes its place. Therefore the new context has length $c = c - 1$. Each candidate son of `Bygone` is subject to the `RowFutureBleakQ` test

[8] Of course, cycle partitions may be interesting in their own right, e.g. as decent substitutes for non-existing Hamiltonian cycles. This is akin to polyhedral combinatorics where cycle partitions are called 2-factors and used as relaxations of the Travelling Salesman problem [6, Chapters 30, 58].

Table 4
The (d, g)-algorithm

Input: A connected (v, w)-graph G
Output: All Hamiltonian cycles of G, stored as 0, 1-vectors row[1], ..., row[c].

```
row[1] = (2, 2, ..., 2)  (* it has length w *)
c = 1          (* in general c gives the length of the actual context *)
Do[ p = c       (* the main loop processess arbitrary v − 1 vertices *)
   While [ p > 0,
⋮    ⋮          If[¬ RowFutureBleakQ[row[p]],
⋮    ⋮            q = |star(α_i) ∩ con[row[p]]|
⋮    ⋮            If[q == 0, Row = TrivialSon[row[p]]
⋮    ⋮                       If[¬ RowFutureBleakQ[Row], row[p] = Row
⋮    ⋮                                                , row[p] = row[c]
⋮    ⋮                                                 c = c − 1]
⋮    ⋮                     , (* now comes the case q > 0 *)
⋮    ⋮                       Bygone = row[p]
⋮    ⋮                       row[p] = row[c]
⋮    ⋮                       c = c − 1
⋮    ⋮                       Do[TryMe = CandidateSon[Bygone, k]
⋮    ⋮                          If[¬RowFutureBleakQ[TryMe], c = c + 1
⋮    ⋮                                                      row[c] = TryMe]
⋮    ⋮                     , {k, 1 + q + (q/2)}  ]
⋮    ⋮               ]
⋮    ⋮          ]
⋮       p = p − 1  ]
,  {i, v − 1}  ]
```

Table 5

(v, w)	hamcyc(G)	(d, g)-alg.	Mathematica command	vdcpart(G)	(d, g)-alg. (without RowFutureBleakQ)
(10, 23)	72	0.2	3.7	131	0.2
(10, 34)	8292	22.6	224.6	13165	18.7
(10, 37)	16728	43.8	600.1	26699	35.2
(30, 67)	35840	416.7	20147.2	88513	446.6
(300, 1353)	0	0.1	26.1	0	0.1

and, if it passes, is appended (possibly in altered form) at the *end* of the context, triggering $c = c + 1$. The complete Mathematica code can be obtained from the author upon request (send an email).

With hamcyc(G) and vdcpart(G) being the number of Hamiltonian cycles, respectively cycle partitions of G, Table 5 displays some experimental data. The (10, 23), (10, 34), and (30, 67)-graphs are identical to the ones in Section 4. The (d, g)-algorithm prevails throughout over Mathematica's HamiltonianCycle[G,All], in particular notice the 0.1 versus 26.1 seconds for the large graph with *no* Hamiltonian cycle. Similar to the relation between cyc(G) and vdcpack(G) in Table 2, here vdcpart(G) stays within a factor 3 of hamcyc(G). For instance,

$88513 < 3 \cdot 35840$. It happened that vdcpart(G) was faster computed than hamcyc(G) (e.g. 18.7 sec $<$ 22.6 sec), but even then the combined time of first producing all cycle partitions and *afterwards* sieving all Hamiltonian cycles (with Mathematica's `ConnectedQ`), was considerably more. For instance, for the (10, 34)-graph the figures are 22.6 versus $18.7 + 16.8$ seconds.

The (d, g)-algorithm may be bent in several ways. Let us sketch a few. The *directed* Hamiltonian cycles in a digraph [3, Chapters 5,6] can be handled by demanding $\deg_{\text{in}}(X, \alpha) = \deg_{\text{out}}(X, \alpha) = 1$ instead of (4). Hamiltonian cycles X (directed or undirected) satisfying a plethora of extra conditions still fit the principle of exclusion: Forcing arbitrary edges to be contained in X is as easy as inserting a couple of 1's in the initial context C_0, and imposing "implications" of type $(\{x_1, \ldots x_m\} \subseteq X \Rightarrow \{y_1, \ldots, y_n\} \subseteq X)$ is not much harder. Finding *one* Hamiltonian cycle, or finding an optimal Travelling Salesman Tour from a provided (e.g. by linear programming) near-optimal target value, are other tempting projects. We devote a whole section to yet another topic, mainly to further illustrate the `RowFutureBleakQ` subroutine.

6. All Hamiltonian paths

We fix any two vertices of G, call them α and β and want to generate all Hamiltonian paths X from α to β. Identifying X with its underlying edge set, it is easily seen that this amounts to find all $X \subseteq E$ that satisfy[9]

$$\deg(X, \alpha) = \deg(X, \beta) = 1, \qquad \deg(X, \gamma) = 2 \quad \text{for all } \gamma \in V \setminus \{\alpha, \beta\}, \tag{5}$$

$$|X| \geqslant v - 1. \tag{6}$$

As opposed to (4), the connectedness of X presents no problem here. Let us generate all Hamiltonian paths from $\alpha = \alpha_2$ to $\beta = \alpha_5$ in our toy graph G. None of them will be part of a Hamiltonian cycle since there is no edge between α and β. The context C_1 in Table 6 features the boundary conditions $\deg(X, \alpha) = \deg(X, \beta) = 1$, and the other constraints $\deg(X, \gamma) = 2$ are imposed as in Section 4, say in order $\alpha_1, \alpha_3, \alpha_4, \alpha_6$. Some more details. Upon processing α_3 we get C_3' from C_2. Applying the subroutine `RowFutureBleakQ` to r_1, r_2 yields twice `False` and, glimpsing at α_4, returns the proper sons $r_1(+)$, $r_2(+)$. As to the third row in C_3', `RowFutureBleakQ` (that is to say 0, 1-propagation) does not catch on here, and so $r_3(+) = r_3$ is copied unaltered into C_3. Subjecting the unsplit son s_1 of

Table 6

	1	2	3	4	5	6	7	8	9	10	
	2	2	2	g_1	g_1	g_1	2	2	g_2	g_2	C_1
r	d	d	d	**0**	g_1	g_1	2	2	g_2	g_2	C_2
s	g_1	g_1	g_1	**1**	0	0	2	2	g_2	g_2	
r_1	g_1	g_1	**1**	0	1	**0**	1	2	g_2	g_2	
r_2	1	1	**0**	0	0	**1**	1	2	g	g	C_3'
r_3	g_1	g_1	**1**	0	0	**1**	0	2	g_2	g_2	
s_1	0	0	1	1	0	0	1	2	g	g	
$r_1(+)$	1	0	1	0	1	0	1	0	0	1	
$r_2(+)$	1	1	0	0	0	1	1	0	0	1	C_3
$r_3(+)$	g_1	g_1	1	0	0	1	0	2	g_2	g_2	
$s_1(+)$	0	0	1	1	0	0	1	1	0	1	
	1	0	1	0	1	0	1	0	0	1	
	1	1	0	0	0	1	1	0	0	1	
t_1	1	**0**	1	0	0	1	0	2	**0**	1	
t_2	1	**0**	1	0	0	1	0	1	**1**	0	C_4'
t_3	0	**1**	1	0	0	1	0	1	**0**	1	
t_4	0	**1**	1	0	0	1	0	0	**1**	0	
	0	0	1	1	0	0	1	1	0	1	

[9] In view of (6) the equalities $= 1$ and $= 2$ in (5) could be replaced by the inequalities $\leqslant 1$ and $\leqslant 2$. As argued for matchings in Section 7a of [9], it follows that the simplicial complex $\mathcal{J}$ generated by the Hamiltonian α, β-paths can be enumerated in time $O(|\mathcal{J}|v^2 w)$.

C_3' to `RowFutureBleakQ` yields (due to α_6) the row $s_1(+)$ of C_3. Upon processing $\operatorname{star}(\alpha_4) = \{2, 5, 7, 8, 9\}$ the rows in $C_3 \setminus \{r_3\}$ carry over unaltered to C_4' but $t := r_3$ splits. Putting $q = |\operatorname{con}(t) \cap \operatorname{star}(\alpha_4)| = 2$, two of the $1 + q + \binom{q}{2} = 4$ (pre-)candidate sons t_1 to t_4 of t won't make it. Namely, since $\operatorname{star}(\alpha_4)$ restricted to t "is" $\{g_1, 0, 0, 2, g_2\}$, we cannot afford to switch both g_1 and g_2 to $\mathbf{0}$ in t_1. Otherwise $\{0, 0, 0, 2, 0\}$ does not contain enough 1's and 2's to meet the requirement $\deg(X, \alpha_4) = 2$! Actually, our implementation of the (d, g)-algorithm is fine-tuned enough not to generate this kind of pre-candidate son in the first place. As to t_4, it bites the dust since `RowFutureBleakQ`$[t_4] =$ `True` in preview of $\operatorname{star}(\alpha_6) = \{1, 8, 10\}$. Since processing the last vertex α_6 is, as always, redundant, the final context $C_4 := C_4' - \{t_1, t_4\}$ gives these five Hamiltonian α, β-paths:

$$\{1, 3, 5, 7, 10\}, \quad \{1, 2, 6, 7, 10\}, \quad \{1, 3, 6, 8, 9\}, \quad \{2, 3, 6, 8, 10\}, \quad \{3, 4, 7, 8, 10\}.$$

References

[1] N. Gibbs, A cycle generation algorithm for finite undirected linear graphs, J. Assoc. Comput. Mach. 16 (1969) 564–568.
[2] Z. Jackowski, Methods for determining all the chordless cycles of a bipartite graph and for testing whether a binary matrix is balanced, Arch. Control Sci. 2 (1993) 55–61.
[3] J. Bang-Jensen, G. Gutin, Digraphs: Theory, Algorithms and Applications, Springer-Verlag, London, 2001.
[4] P. Mateti, N. Deo, On algorithms for enumerating all circuits of a graph, SIAM J. Comput. 5 (1976) 90–99.
[5] R.C. Read, R.E. Tarjan, Bounds on backtrack algorithms for listing cycles, paths, and spanning trees, Networks 5 (1975) 237–252.
[6] A. Schrijver, Combinatorial Optimization, Springer-Verlag, Berlin, 2003.
[7] M. Syslo, An efficient cycle vector space algorithm for listing all cycles of a planar graph, SIAM J. Comput. 10 (1981) 797–808.
[8] J. Welch, A mechanical analysis of the cyclic structure of undirected linear graphs, J. Assoc. Comput. Mech. 13 (1966) 205–210.
[9] M. Wild, Generating all perfect matchings of a not necessarily bipartite graph, in preparation.